Query, Key, Values

All attention = query, Key, Value

Q, K, V is learned

Bank is very cute and he is a dog

V_4 = weighted (V_1, V_2, V_3)

I love V and regardless what K, Q, I love should overwhelmingly of V_i can be influenced by key/query value

$[1 \times n] * [n \times n] = [1 \times n]$

$\text{weighted sum} = [n \times n]$

$w_3 = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$

$w_3 = \sum_{i=1}^{n} v_i$

I don't care the dim by hand attention

Attention in detail

Attention $\{q_1, v_1\}$ $\{v_2\}$, Output = query q_1 and key k_1

Attention (q, k, v) = $\sum$ (similarity q, k) v_i

1. Similarity q, k gives k_1
2. Similarity is weight
3. weighted sum of all values = output

The only difference between database retrieval and attention in a sense is that in database retrieval we only get one value as input, but here we get a weighted combination of values.

Step 1 :- Query and Key Similarity

→ n-gram similarity

$S_1 = f(q_1, k_1) = q_1^T k_1$ ..... dot product

$= q_1^T k_1 / \sqrt{d}$ ..... scaled dot product (advantageous gradient)

$= q_1^T U k_1$ ..... general dot product (query weight vs. key weight, learned)

:- learned similarity

Step 2 :- Softmax weights (n)-different attention heads

$S_n = \frac{\exp(S_n)}{\sum_j \exp(S_j)}$

Step 3 :- weighted sum of softmax(n) and weighted values

attention value $= \sum_n S_n V_n$

Multi Head Attention

Multi head attention
↳ 8 different subspaces of dot product attention
"Multi-head attention"

Our words can mean different things at same time depending on context

Dog word in front of a dog
a dog walk (in)
he is cute

Eg. 8 attention will be different heads

So let's add more linear layers as the keys, queries, and values. These linear layers are training in parallel, and have independent weights to one another.

But it is just a random number as shows. In the actual scenario, these can be the different subspaces and these are called heads. (h)

Transformer

Transformer - Reflected RNNs for NLP

- I hopefully Vector = Vector transformers

→ Decoder → to produce translation sentence

Encoder
↓
to encode input sentence

Can generate words when to predict together the h_1 h2 through encoder

Encoder Block

PE : Without positional embedding, the model will treat the entire sentence as just a bag full of words without having any sequence or meaning

Input Embedding → takes words into d-vector

Input Embedding → The word 'dog' in a sentence can use the embedding space to obtain a vector embedding. Embedding is just converting a word in any language to its vector representation.

In the embedding space, similar words have similar embeddings, for example, the word 'cat' and the word 'kitty' will fall very close in the embedding space, and the word 'car' and 'wooden' will fall further away in the space.

Positional Embedding → A word in different sentences can have different meanings. For example, the word dog in a 1 over a cute dog (emotional — position 5) and 5. What a lazy dog you are! (worthless -position 4) have different meanings.

→ use PE helps
→ similar other given sentence changes → contextual word embedding
Q word → before
→ Transformers use PE instead of RNN to embed sequence. Instead, PE turns PE to embedding to embedding sequence.

⇒ This is sort of a vector encoding defining the position of a word.

→ This positional embedding is added to the input embedding before going into the attention network.

Input Embedding		Positional Embedding
0.33		0.02
0.71	+	0.21
0.91		0.33
0.23		0.43

After the positional embedding and positional embedding is complete, the embeddings then flow into the most important part of the encoder which contains the two important blocks ...

1. a multi-head attention block and
2. A feed-forward network.

Multi head Attention → which weight coefficient

As input, this block receives a vector (the sentence) that contains a sub-vector (words in a sentence). The multi-head attention then computes the attention between every position with every other position of the vector.

→ 3 word embedding added → self attention

Scale : $\frac{1}{\sqrt{d}}$

Output with Scale = $\left(\frac{Q^T K}{\sqrt{d}}\right) V$

Q, K → similarity

$\left(\frac{Q^T K}{\sqrt{d}}\right)$ → d → independently of other embeddings (Exp)

The word embeddings are first passed into some linear layers. These linear layers do not have a 'bias' term, and hence are nothing but matrix multiplications.

In simple words, the idea of multi-head attention is to take a word embedding, combine it with some other word embedding (or multiple words) using attention (or multiple attentions) to produce a better embedding for that word (embedding with a bit more context of the surrounding words).

The idea is to compute multiple attentions per query with different weights.

Add & Norm and Feed Forward

The next block is the 'Add & Norm' which takes in a residual connection of the original word embedding, adds it to the embedding from the multi-head attention, and then normalizes it to have zero mean and variance 1.

This is fed to a 'feed forward' block which also has an 'add & norm' block at its output.

The whole multi-head attention and feed-forward blocks are repeated n times (hyperparameter), in the encoder block.

Output of Encoder → one embedding per position, where each position embedding contains not only the embedding of the original word at the position but also information about other words, that it learned using attention.

Decoder Block

Output embedding from Encoder

The purpose of the decoder is to produce some output. In the paper, Attention is All You Need, the decoder was used for sentence translation (say from English to French). So the encoder will take in the English sentence, and the decoder will translate it to French.

Steps in decoder

Input from encoder, Output embedding through attention block

Positional Encoding of English word in French sentence translation

English - French Attention Vectors translation (English to French output)

Self attention of the French words

Mask embedding and positional encoding for the French vectors

French

Masked Multi-head Attention

This is a multi-head attention block where some values are masked. The probabilities of the masked values are nullified or not selected.

For example, while decoding, the output value should only depend on previous outputs and not future outputs. Then we mask the future outputs.

$$\text{Attention}(Q,K,V) = \text{SoftMax}\left(\frac{QK_T}{\sqrt{d_k}}\right)V$$

$$\text{Masked Attention}(Q,K,V) = \text{SoftMax}\left(\frac{QK_T + M}{\sqrt{d_k}}\right)V$$

where M is a mask matrix of 0's and $-\infty$'s